Corso di laurea in Informatica
Progettazione d'algoritmi

Tecnica Divide et Impera 2

Angelo Monti

SAPIENZA
UNIVERSITÀ DI ROMA

Il problema della selezione

Data una lista A di interi distinti, il rango di un elemento x in A è il numero di elementi che sono minori o uguali ad x in A.

Esempi:

- il minimo in A è l'elemento di rango 1.
- il mediano in A è l'elemento di rango $\lceil \frac{n}{2} \rceil$
- il massimo in A è l'elemento di rango n.

Problema: Abbiamo una lista A di n numeri distinti ed un intero k, $1 \leq k \leq n$. Vogliamo l'elemento di rango k in A.

Un semplice algoritmo di complessità $\Theta(n \log n)$ è il seguente:

```
def selezione1(A,k):
    A.sort()
    return(A[k-1])

>>> A=[30,20,50,60,10,40,90,80,70]
>>> selezione1(A,4)
40
```

- Se $k = 1$ o $k = n$ il problema si riduce alla ricerca del *minimo* o del *massimo* e questi casi sono risolvibili in tempo $\Theta(n)$.
- Utilizzando la tecnica del *divide_et_impera*, vedremo che, anche nel caso generale, il problema può risolversi in $\Theta(n)$.
- In altre parole dimostreremo che il problema della selezione è computazionalmente più semplice di quello dell'ordinamento (che richiede tempo $\Omega(n \log n)$).

Approccio basato sul Divide et impera:

- scegli nella lista A un elemento x (che chiameremo perno).
- a partire da A costruisci due liste A_1 ed A_2, la prima contenente gli elementi di A minori di x e la seconda gli elementi di A maggiori di x.
- dove si trova l'elemento di rango k?
  - se $|A_1| \geq k$ allora l'elemento di rango k è nel vettore A_1
  - se $|A_1| = k - 1$ allora l'elemento di rango k è proprio il perno x.
  - se $|A_1| < k - 1$ allora l'elemento di rango k è in A_2 è l'elemento di rango $k - |A_1| - 1$ in A_2

Nota che: dopo aver costruito dalla lista A di n elementi le due liste $A1$ e $A2$, grazie al test sulla cardinalità di $A1$ il problema della selezione dell'elemento di rango k o risulta risolto o viene ricondotto alla selezione di un elemento in una lista con meno di n elementi.

```
def selezione2(A,k):
    #restituisce l'elemento di rango k dove 1<=k<=len(A)
    if len(A)==1:return A[0]
    perno=A[0]
    lista1,lista2=[],[]
    for x in A:
        if x<perno:
            lista1.append(x)
        elif x>perno:
            lista2.append(x)
    if len(lista1)>=k:
        return selezione2(lista1,k)
    elif len(lista1)==k-1:
        return perno
    return selezione2(lista2,k-len(lista1)-1)
```



```
def selezione2(A,k):
    #restituisce l'elemento di rango k dove 1<=k<=len(A)
    if len(A)==1:return A[0]
    perno=A[0]
    lista1,lista2=[],[]
    for x in A:
        if x<perno:
            lista1.append(x)
        elif x>perno:
            lista2.append(x)
    if len(lista1)>=k:
        return selezione2(lista1,k)
    elif len(lista1)==k-1:
        return perno
    return selezione2(lista2,k-len(lista1)-1)
```

Complessità al caso peggiore: Possiamo limitare la complessità dell'algoritmo con la seguente ricorrenza

$$T(n) \leq T(m) + \Theta(n)$$

dove m è la cardinalità massima tra quella delle due sottoliste $lista1$ e $lista2$ generate.

La procedura che tripartisce la lista in $lista1$, p e $lista2$ può restituire una partizione massimamente sbilanciata in cui si ha ad esempio

$$|lista1| = 0 \quad \text{e} \quad |lista2| = n - 1.$$

questo accade quando il perno p scelto risulta l'elemento minimo nella lista. Qualora questo evento sfortunato si ripesse sistematicamente nel corso delle varie partizioni eseguite dall'algoritmo (questo può accadere quando cerco il massimo in una lista ordinata), allora la complessità dell'algoritmo al caso pessimo è catturata dalla seguente equazione:

$$T(n) = T(n-1) + \Theta(n)$$

che risolta dà $T(n) = \Theta(n^2)$

Se avessimo una regola di scelta del perno p in grado di garantire una partizione perfettamente bilanciata (vale a dire $m = \max\{|lista1|, |lista2|\} = n/2$) allora per la complessità $T(n)$ dell'algoritmo avremmo

$$T(n) = T\left(\frac{n}{2}\right) + \Theta(n)$$

che produce

$$T(n) = O(n)$$

Chiedere che la partizione sia perfettamente bilanciata forse è chiedere troppo, potremmo allora accontentarci di chiedere che p garantisca partizioni non troppo sbilanciate come ad esempio quelle per cui $m \leq 3n/4$. In questo caso si ha

$$T(n) \leq T\left(\frac{3}{4}n\right) + \Theta(n)$$

che risolta dà ancora $T(n) = O(n)$.
In generale finché m è una frazione di n (anche piuttosto vicina ad n come ad esempio $\frac{99}{100}n$) la ricorrenza dà sempre $T(n) = O(n)$.

IDEA:
scegliamo il perno p a caso in modo equiprobabile tra gli elementi della lista.

- Anche se la scelta "casuale" non produce necessariamente una partizione bilanciata, quanto visto ci fa intuire che la complessità rimane lineare in n.

```
import random

def selezione2R(A,k):
    #restituisce l'elemento di rango k dove 1<=k<=len(A)
    if len(A)==1:return A[0]
    perno=A[random.randint(0,len(A)-1)]
    lista1,lista2=[],[]
    for x in A:
        if x<perno:
            lista1.append(x)
        elif x>perno:
            lista2.append(x)
    if len(lista1)>=k:
        return selezione2R(lista1,k)
    elif len(lista1)==k-1:
```


Più formalmente, sotto l'assunzione che il pivot possa essere con uguale probabilità un elemento di qualsiasi rango tra 1 e n la ricorrenza da studiare per il caso medio è

$$T(n) \leq \frac{1}{n}\sum_{k=1}^{n} T(\max(T(k-1), T(n-k))) + \Theta(n) = \frac{1}{n}\sum_{k=\lceil \frac{n}{2} \rceil}^{n-1} 2T(k) + \Theta(n) \text{ per } n \geq 2$$

possiamo dimostrare che per questa ricorrenza vale T(n)=O(n) col metodo di sostituzione.

Sia $T(n) \leq \frac{1}{n}\sum_{k=\lceil \frac{n}{2} \rceil}^{n-1} 2T(k) + a \cdot n$ per $n \geq 2$, $T(1) \leq b$ e dimostriamo $T(n) \leq cn$ per una c costante.
Per $n = 2$ abbiamo $T(2) \leq 2b + 2a \leq c$ che è vera per c sufficientemente grande.
Per ipotesi induttiva abbiamo $T(n) \leq \frac{2c}{n}\sum_{k=\lceil \frac{n}{2} \rceil}^{n-1} k + a \cdot n$ da cui ricaviamo

$$T(n) \leq \frac{2c}{n}\left(\sum_{k=1}^{n-1} k - \sum_{k=1}^{\lceil \frac{n}{2} \rceil - 1} k\right) + a \cdot n \leq \frac{2c}{n}\left(\frac{n(n-1)}{2} - \frac{(n/2)(n/2-1)}{2}\right) + a \cdot n \leq \frac{c}{n}(n^2 - n - n^2/4 + n/2) + a \cdot n \leq \frac{3}{4}cn + an \leq cn$$

dove l'ultima diseguaglianza segue prendendo $c \geq 4a$

Algoritmo probabilistico per la selezione:

```
import random

def selezione2R(A,k):
    #restituisce l'elemento di rango k dove 1<=k<=len(A)
    if len(A)==1:return A[0]
    perno=A[random.randint(0,len(A)-1)]
    lista1,lista2=[],[]
    for x in A:
        if x<perno:
            lista1.append(x)
        elif x>perno:
            lista2.append(x)
    if len(lista1)>=k:
        return selezione2R(lista1,k)
    elif len(lista1)==k-1:
```

- L'analisi rigorosa appena fatta dimostra che, se la scelta del perno avviene in modo equiprobabile a caso tra i vari elementi della lista A, il tempo di calcolo dell'algoritmo risulta con alta probabilità lineare in n.
- il tempo dell'algoritmo *selezione2R* è dunque $O(n)$ con alta probabilità.
- Ovviamente nel caso peggiore, quando nelle varie partizioni che si succedono nell'iterazione dell'algoritmo si verifica che il perno scelto a caso risulta sempre vicino al massimo o al minimo della lista, la complessità dell'algoritmo rimane $O(n^2)$. Però questo accade con probabilità molto piccola.

Vedremo ora un algoritmo deterministico che garantisce complessità $O(n)$ anche al caso pessimo.

Sappiamo che riuscire a selezionare un perno in grado di garantire che nessuna delle due sottoliste $lista1$ e $lista2$ abbia più di cn elementi per una qualche costante $0 < c < 1$, avrebbe come conseguenza una complessità di calcolo $O(n)$

Descriviamo ora un metodo (noto come il mediano dei mediani) per selezionare un perno p che garantisce di produrre **sempre** due sottoliste $lista1$ ed $lista2$ ciascuna delle quali ha al più $\frac{3}{4}n$ elementi:

- **Dividi gli n elementi di $lista$ in gruppi di 5 elementi ciascuno tranne al più l'ultimo che potrebbe averne di meno e considera i primi $\lfloor \frac{n}{5} \rfloor$ gruppi ottenuti (tutti di 5 elementi ciascuno)**
- **Trova il mediano in ciascuno di questi $\lfloor \frac{n}{5} \rfloor$ gruppi.**
- **Trova il mediano p dei $\lfloor \frac{n}{5} \rfloor$ mediani.**
- **Usa p come perno di $lista$.**

Vedremo ora un algoritmo deterministico che garantisce complessità $O(n)$ anche al caso pessimo.

- **Dividi gli n elementi di A in gruppi di 5 elementi ciascuno tranne al più l'ultimo che potrebbe averne di meno e considera i primi $\lfloor \frac{n}{5} \rfloor$ gruppi ottenuti (tutti di 5 elementi ciascuno)**
- **Trova il mediano in ciascuno di questi $\lfloor \frac{n}{5} \rfloor$ gruppi.**
- **Trova il mediano p dei $\lfloor \frac{n}{5} \rfloor$ mediani.**
- **Usa p come perno di A.**

ESEMPIO:

A=[15, 2, 10, 16, 21,12, 1, 9, 11, 17, 22, 3, 8,13, 14, 4, 9, 19, 5, 6, 20, 23,18, 7]

15, 2, 10, 16, 21 | 12, 1, 9, 11, 17 | 22, 3, 8, 13, 14 | 4, 19, 5, 6, 20 | 23, 18, 7

2,10, 15, 16, 21 | 1, 9, 11,12, 17 | 3, 8, 13, 14, 22 | 4, 5, 6, 19, 20 |

15, 11, 13, 6

6, 11, 13, 15

p=11

PROPRIETA': Se la lista A contiene almeno 120 elementi e il perno con cui partizionarla viene scelto in base alla regola appena descritta si può esser sicuri che la dimensione di ciascuna delle due sottoliste $lista1$ e $lista2$ ottenute sarà limitata da $\frac{3}{4}n$.

PROVA:

- il perno scelto p ha la proprietà di trovarsi in posizione $\lceil \frac{n}{10} \rceil$ nella lista degli $\lfloor \frac{n}{5} \rfloor$ mediani selezionati in $lista$.
  Ci sono dunque $\lceil \frac{n}{10} \rceil - 1$ mediani di valore inferiore a p e $\lfloor \frac{n}{5} \rfloor - \lceil \frac{n}{10} \rceil$ mediani di valore superiore a p.
- Prova che $|lista2| < \frac{3}{4}n$:
  - considera gli $\lceil \frac{n}{10} \rceil - 1$ mediani di valore inferiore a p.
  - Ognuno di questi mediani appartiene ad un gruppo di 5 elementi in n.
  - Ci sono dunque in n altri 2 elementi inferiori a p per ogni mediano. In totale abbiamo $3\left(\lceil \frac{n}{10} \rceil - 1\right) \geq 3\frac{n}{10} - 3$ elementi di $lista$ che finiranno in $lista1$.
  - $|lista2| \leq n - (3\frac{n}{10} - 3) = \frac{7}{10}n + 3 \leq \frac{3}{4}n$ (dove l'ultima diseguaglianza segue dal fatto che $n \geq 120$)
- Prova che $|lista1| < \frac{3}{4}n$:
  - $\lfloor \frac{n}{5} \rfloor - \lceil \frac{n}{10} \rceil \geq \left(\frac{n}{5} - 1\right) - \left(\frac{n}{10} + 1\right) = \frac{n}{10} - 2$ mediani di valore superiore a p.
  - Ognuno di questi mediani appartiene ad un gruppo di 5 elementi in n.
  - Ci sono dunque in n altri 2 elementi superiori a p per ogni mediano. In totale abbiamo almeno $3\frac{n}{10} - 6$ elementi di $lista$ che finiranno in $lista2$.
  - $|lista1| \leq n - (3\frac{n}{10} - 6) = \frac{7}{10}n + 6 \leq \frac{3}{4}n$ (dove l'ultima diseguaglianza segue dal fatto che $n \geq 120$)

Implementazione dell'algoritmo selezione:

```
from math import ceil

def seleziane(A,k):
    # restituisce l'elemento di rango k dove 1<=k<=len(A)
    # scegliendo ogni volta il perno con la regola del mediano dei mediani
    if len(A)<= 120:
        A.sort()
        return A[k-1]
    #crea una lista che contine i primi
    # inizializza B con i mediani dei len(A)//5 gruppetti di 5 elementi di A
    B=[sorted(A[5*i:5*i+5])[2] for i in range(len(A)//5)]
    #individua il perno p con la regola del mediano dei mediani
    perno=selezione(B,ceil(len(A)/10))
    for x in A:
        if x<perno: lista1.append(x)
        elif x>perno:lista2.append(x)
    if len(lista1)>=k:
        return selezione(lista1,k)
    elif len(lista1)==k-1:
        return perno
    return selezione(lista2,k-len(lista1)-1)
```

Implementazione dell'algoritmo selezione:

```
from math import ceil

def seleziane(A,k):
    # restituisce l'elemento di rango k dove 1<=k<=len(A)
    # scegliendo ogni volta il perno con la regola del mediano dei mediani
    if len(A)<= 120:
        A.sort()
        return A[k-1]
    #crea una lista che contine i primi
    # inizializza B con i mediani dei len(A)//5 gruppetti di 5 elementi di A
    B=[sorted(A[5*i:5*i+5])[2] for i in range(len(A)//5)]
    #individua il perno p con la regola del mediano dei mediani
    perno=selezione(B,ceil(len(A)/10))
    for x in A:
        if x<perno: lista1.append(x)
        elif x>perno:lista2.append(x)
    if len(lista1)>=k:
        return selezione(lista1,k)
    elif len(lista1)==k-1:
        return perno
    return selezione(lista2,k-len(lista1)-1)
```

Complessità:

- nota che:
  - ordinare 120 elementi richiede tempo $O(1)$.
  - ordinare una lista di n elementi in gruppetti da 5 richiede tempo $\Theta(n)$.
  - selezionare i mediani dei mediani di gruppi da 5 da una lista in cui gli elementi sono stati ordinati in gruppetti da 5 richiede tempo $\Theta(n)$.

- Sappiamo che per $n \geq 120$ risulta $|lista1| \leq \frac{3}{4}n$ e $|lista2| \leq \frac{3}{4}n$.
  Dunque per la complessità $T(n)$ dell'algoritmo si ha

$$T(n) \leq \begin{cases} O(1) & \text{se } n \leq 120 \\ T\left(\frac{n}{5}\right) + T\left(\frac{3n}{4}\right) + \Theta(n) & \text{altrimenti} \end{cases}$$

Notiamo che la ricorrenza è del tipo

$T(n) = T(\alpha \cdot n) + T(\beta \cdot n) + \Theta(n)$ con $\alpha + \beta = 1/5 + 3/4 = 19/20 < 1$.

mostreremo nel prossimo lucido che ricorrenze di questo tipo hanno tutte

come soluzione

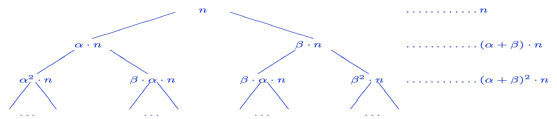
$$T(n) = \Theta(n).$$

ESERCIZIO:

Se $T(n) = T(\alpha \cdot n) + T(\beta \cdot n) + cn$ e $\alpha + \beta < 1$, allora $T(n) = \Theta(n)$.

PROVA:

- Il fatto che $\alpha + \beta$ sia inferiore ad 1 gioca un ruolo fondamentale nella prova.
  Consideriamo l'albero delle chiamate ricorsive generato dalla ricorrenza e analizziamone il costo per livelli.



- Al primo livello abbiamo un costo $(\alpha+\beta)\cdot n$, al secondo un costo $(\alpha+\beta)^2 \cdot n$ al terzo un costo $(\alpha + \beta)^3 \cdot n$ e così via.

- Il tempo di esecuzione totale è la somma dei contributi dei vari livelli:

$$T(n) < c \cdot n + c \cdot (\alpha+\beta) \cdot n + c \cdot (\alpha+\beta)^2 \cdot n \ldots = c \cdot n \cdot \sum_{i=0}^{\infty} (\alpha+\beta)^i = c \cdot n \frac{1}{1-(\alpha+\beta)} = \Theta(n)$$

dove nel calcolare la serie abbiamo sfruttato il fatto che $\alpha + \beta < 1$ e la serie geometrica $\sum_{i=0}^{\infty} x^i$, con $x < 1$, converge a $\frac{1}{1-x}$.

Abbiamo quindi dimostrato che il problema della selezione può essere risolto in tempo lineare.
Abbiamo infatti un algoritmo che risolve il problema in $O(n)$ al caso pessimo.
Tuttavia, a causa delle grandi costanti moltiplicative nascoste dall'$O(n)$, nella pratica l'algoritmo randomizzato che ha tempo $O(n)$ con alta probabilità si comporta molto meglio.